DOCUMENT RESUME

ED 258 552 IR 011 707

AUTHOR	Sleeman, D.; And Others
TITLE	Pascal and High-School Students: A Study of Misconceptions. Technology Panel Study of Stanford and the Schools. Occasional Report #009.
INSTITUTION	Stanford Univ., Calif. School of Education.
PUB DATE	84
NOTE	; For related document, see IR 011 711.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC02 Plus Postage.
DESCRIPTORS	Course Evaluation; *Error Patterns; *Experimental Teaching; High Schools; High School Students; *Instructional Material Evaluation; Learning Experience; *Programing; Programing Languages; Psychological Studies; Screening Tests; Teaching Methods; Test Construction; Test Results
IDENTIFIERS	*Pascal Programing Language

ABSTRACT

In an attempt to initiate a new approach to the teaching of Pascal, a study was conducted to ascertain the difficulties students encountered when they attempt to learn this computer language. Screening tests were given to 68 students in grades 11 and 12 who had just completed a semester course in Pascal. The purpose of the test was to detect possible difficulties in basic constructs such as reading and printing data, assignments, and the several control structures provided by Pascal. This test that showed over 50% of students had major difficulties with Pascal, and those problems are described, with notations as to whether the errors were frequent, or fairly frequent, or occasional. A group of 35 students were given a detailed clinical interview which is also described, and their explanations of why errors occurred are given as well. A discussion of the data includes a summary of the investigators' assessment of the students interviewed and profiles of typical students. Finally, a comparison is drawn between this and similar studies, and the report concludes with a plan for future investigations which will include: (1) a look at the difficulties high school students have with advanced concepts of Pascal and also with Logo; (2) an attempt to determine whether the errors noted in this study can be remediated; and (3) experimentation with different teaching/presentation strategies. (JB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED258552

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

OCCASIONAL REPORT # 009

Technology Panel

Study of Stanford and the Schools

Pascal and High-School Students:

A Study of Misconceptions

D. Sleeman
Ralph T. Putnam
Juliet A. Baxter
Laiani K. Kuspa

August 1984

These occasional reports present preliminary findings of research underway or discuss issues of concern to the panel. They are intended to stimulate comment and to maintain communication with interested parties both within and outside the Study of Stanford and the Schools. They are draft documents and are not to be quoted or cited. They do not necessarily represent the views of Stanford, the Study as a whole, or even the panel as a whole. Members of the panel and addresses where they may be reached are included at the rear of the report. Comments of readers will be greatly appreciated.

BEST COPY AVAILABLE

IR011707

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Derek Sleeman
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Pascal and High-School Students:

A Study of Misconceptions

D. Sleeman[1]
Ralph T. Putnam
Juliet A. Baxter
Laiani K. Kuspa

School of Education
Stanford, University
Stanford, CA 94305

Abstract

A screening test was given to 3 classes of high school students, who were just completing introductory semester-long courses in Pascal. These tests were graded, and subsequently 35 students were given detailed clinical interviews. These interviews showed that errors were made with essentially every Pascal construct. Over half the students were classified as having major difficulties -- less than 10% had no difficulties.

The errors noted are discussed in detail in this paper. A major finding is that the students attribute to the computer the reasoning power of an average human. The paper also speculates about how difficult it might be to remediate the errors found, and concludes with an outline of future work.

1. Introduction

The growing availability and use of computers in the past few years has resulted in the introduction of programming courses in many schools. High Schools offer instruction in programming on the grounds that it provides students with needed job skills, that it is an important component of computer literacy, or that it is a powerful way to develop problem-solving and analytical thinking skills. Because programming, particularly in the high school curriculum, is a relatively new phenomenon, we have a limited understanding of how these students learn to program -- the difficulties they have and the misconceptions they develop. Understanding these issues should serve an important role in improving instruction in this area as well as providing insight on the more general area of the learning of complex skills.

It is widely accepted that to program effectively one must:

- have a good knowledge of the syntax and semantics of the target programming language. (i.e. have an understanding of the conceptual machine supported by the programming language)
- be able to debug programs
- be able to analyze (complex) tasks and design algorithms aimed at solving these tasks.

The ability to understand (or "read") programs is a by-product of the first two stages. Further, it is generally agreed that the topics are listed above in their order of complexity. That is, task analysis and algorithm design are the most demanding. Several researchers, including Sheil (1981), look at this issue somewhat differently. While not necessarily disagreeing with the "accepted" wisdom, Sheil argues that when expert programmers are given a new task, they retrieve appropriate chunks (such as a loop to add N numbers) from long-term memory and modify these to solve the current task, (Shneiderman, 1976). Sheil argues that teachers of programming should be concerned about how to get novice programmers to this state, suggesting that the traditional approach may not be appropriate. In several other areas, e.g. Physics and Mathematics, Cognitive Science researchers have noted that the organization of domain knowledge in experts is radically different from that of novices (see for instance Gentner & Stevens, 1983). The key educational question is how to produce instructional sequences to effectively convey the content and structure of expert knowledge to novices. It is reasonable to suppose that Sheil would favour giving students complete programs to read, analyze and modify, and for each of these programs to contain one or more useful "chunks". (The detail of the language's syntax and semantics would then be introduced as secondary issues.) This is not a totally new idea (see Bork, 1971), and has been partially included in many introductory level university programming courses.[2]

Although intuitively we accept Sheil's basic point, in this study we were not able to initiate a new approach to the teaching of Pascal. Moreover we felt that further information about the difficulties experienced by students when taught programming using "traditional" methods would provide additional empirical evidence. When we began our study of programming classes in high schools, we expected to study the three stages noted above. However, it soon became clear that a significant number of students in the classes we studied had significant difficulties with the first stage, and were thus hampered in their attempts to implement and extend programs. The first stage - students' knowledge of syntax and semantics-thus became the focus of this study.

As noted earlier, one major component of learning to program is gaining an understanding of the "virtual machine" (Wegner, 1971) or the "conceptual machine" (Norman, 1983) underlying a particular language -- a working model of how various constructs in a language function (DuBoulay & O'Shea, 1981). A programmer must know, for example, what happens when IF or READLN statements are used in a Pascal program. The current study explores some of the misunderstandings of the conceptual machine that some students hold in the early stages of learning to program in Pascal. We have concentrated primarily on their understanding of fundamental constructs such as variables, assignment, and several control constructs. We have examined to a lesser extent their ability to trace and debug programs.

DuBoulay and O'Shea (1981) provide a fairly comprehensive summary of earlier work on learning to program in commonly used languages, including

ERIC

BASIC, FORTRAN, LOGO, and Pascal. The majority of the studies they reviewed reported the constructs of a particular language with which students had difficulty. Frequently the data provided included:

-- the constructs of the language that were most frequently used;

-- the constructs of the language that were most frequently used incorrectly;

-- the error-proneness of these constructs (where error-proneness is the percent of total instances of a construct in which errors were made);

-- some idea of the persistence of particular errors (where persistence indicates the likelihood of them being removed).

The first detailed analysis of the difficulties users have with Pascal was undertaken by Ripley and Druseikis (1978) who did two studies. In their first study of computer science graduate students, they reported that 64% of the programs submitted were free of both syntactic and semantic errors. In their second study of naive programmers, they reported that 58% of the programs submitted were again free of syntactic and semantic errors. With the latter group, the most common syntax errors were associated with the misuse of the semicolon. The second most common source of error was declaration, because of the restrictions on the ordering of declaration keywords. Missing BEGINS and ENDS were the third most common source of errors; Ripley and Druseikis argued that these errors might be resolved if the language had more specific terminators, e.g. BEGINTHEN, BEGINELSE, ENDTHEN, ENDELSE, etc. to replace the "universal" BEGIN and END.

More recently Soloway and his co-workers have analyzed the errors which university students make with assignment and loop constructs in Pascal. The majority of their analyses have been carried out on programs which have been collected automatically by the operating system, (see Soloway, Ehrlich, Bonar & Greenspan, 1982). More recently they have also used interview techniques to probe students' understanding of the assignment construct, showing that I:=I + 1 and SUM:= SUM +N were viewed as different entities; i.e., the pragmatics of the situation dominated these students' interpretation (Bonar and Soloway, 1982). Another survey showed that 34% of the students believed that the WHILE statement acted like a demon.[3]

With the exception of the study just mentioned, a major omission in all these studies is that they did not determine the nature of the errors associated with various constructs. They presented only the frequencies with which certain constructs were incorrectly used in programs written by novice and sometimes expert programmers. Programmers who made errors were not questioned to determine the nature of their misunderstandings.

Cognitive scientists working in other subject domains have postulated mental models which people hold of various physical and symbolic systems (Gentner & Stevens, 1983; Davis et al, 1978; Larkin et al, 1980). Such studies have relied heavily on interview techniques to reveal the nature of people's misunderstandings. We have applied this methodology to study students' understanding of the conceptual machine of a programming language -- their mental models of the language.

The objective of this experiment was to study the errors which students made in interpreting programs -- and from these begin to understand the misconceptions of novice programmers. As a result of this study we hope to facilitate better teaching of programming -- teaching that avoids or corrects these misconceptions. We also hope this study will provide some valuable insights into how students learn to work skillfully in a complex formal system.

Method

Subjects

Students from three high-school classes participated in the study. A pilot study was done with one class of 27 students; two additional classes of 19 and 22 students respectively were involved. All three classes were introductory courses in Pascal.[4] Our studies took place towards the end of each course. The majority of the students were from grades 11 and 12 and had strong math backgrounds (as there were math prerequisites).

Screening Test

Prior to conducting the Pascal study we had carried out an analogous study of the difficulties students encountered when they learn BASIC, (Putnam et al., in press). For the BASIC study we had developed an effective screening test and a set of more detailed question sheets focusing on particular topics. The task at the beginning of the Pascal study was to refine these tools (the test used is available from the authors). The purpose of the test is to detect possible difficulties in basic constructs such as reading and printing data, assignments, and the several control structures provided by Pascal. Nine items require writing the output produced by short (6 to 14 line) programs, each designed to highlight specific concept(s). One task requires the student to debug a program for which a written description of the intent has been provided. Two items address a similar task, but each program uses a different control structure. The test took between 15 and 35 minutes for the students to complete. Because we asked questions about programs which we had prepared, only the students' reading knowledge of Pascal was tested. It would appear that creating a program would be more complex than understanding a given (short) program, and so we suggest that this test represents a test of minimum competence.

The screening test was pretested with a class of 27 students. Minor changes were made in the test before it was used with 2 additional classes. In general the test and the questions used in the interviews were fine-tuned for each class to reflect the teaching materials used, the order in which concepts had been introduced, etc.

Experimental Procedure

The screening test was given to each student in the three classes. Each student's performance was evaluated by one of the researchers who decided that the student should be interviewed, was a marginal candidate for an interview, or did not need an interview (depending on whether the student had minor or no difficulties, or manifested a well understood set of misunderstandings). Interviews were conducted with 9, 15, and 11 students respectively from the three classes (in the case of the first class it was not logistically possible to interview all the students for whom an interview was suggested). The interviews were clinical in nature, with interviewers using questions and short programs prepared in advance, but also following up with various probes and programs composed on the spot. The goal was to clarify as far as possible the nature and extent of the student's misconceptions about programming concepts.

Students were asked to say what output would be produced by various programs, to trace programs and explain how they work, and to debug short programs. In several cases, students were asked to trace identical programs with different sets of input data. The discussion of a particular topic generally continued until the researcher was able to decide: i) the "precise" nature of the student's error, or ii) that the student had a variety of possible ways of interpreting a construct, or iii) that the student had little knowledge of a particular concept. The interviewer also made a subjective assessment about his or her confidence in this prediction and also noted how consistently the student had manifested the several error(s) (for further details of this overall methodology see Sleeman, in preparation). The standard programs and program fragments used in many of the interviews are available from the authors. Some supplementary items created for individuals are included in the text.

Tape recordings, written notes, and responses generated during the interviews were perused for patterns of errors and misconceptions. As the study was exploratory and qualitative in nature, no quantitative analysis techniques were used. Findings are discussed in the following sections.

Section 2 gives an overview of the errors encountered, with some indication of the frequency of their occurrence. Section 3 gives several summaries of the data - including a discussion of typical students with minor and major difficulties, and a classification of the errors noted. Section 4 compares the results of this experiment with earlier studies. The paper concludes with suggestions for future studies.

2. Summary of Errors Encountered

A comment on error frequency

As noted above, the screening tests were given to 68 students of which 35 were subsequently interviewed. We shall refer to an error as being frequent with this population if it occurs with 25% or more of the interview population (i.e. 8 or more students), fairly frequent if it occurs with 4-7 students, and occasional if it occurs less frequently (i.e. with 1-3 students). Note this figure does not capture the frequency or the consistency with which each error was encountered with individual students; specific comments about these aspects will be interspersed throughout this section.

2.1 Difficulties with READLN statements

Several students had difficulty understanding how a READLN statement assigns values to a variable. Four categories of misconceptions appeared: semantically constrained reads, data read in alphabetic order

ERIC

of the variables, order of declaration determines the order of reading from the file, and multiple-value reads.[5]

Semantically constrained reads [IC1a.1]

Eleven students believed that a READLN statement used with a meaningful variable name causes the program to select a value based on the name's meaning. (Thus given the frequency classification given earlier this is a frequent error). For example, given the following program:[6]

```
PROGRAM B1;
VAR First,Smallest,Largest: INTEGER;
BEGIN
     WRITELN('Enter three numbers');
     READLN(Largest,Smallest,First);
END.
[ 5 10 1 ]
```

The students with this error said that 1 would be read into "smallest", 10 into "largest" and 5 into "first". The majority of these students exhibited this error consistently on this program and in two other programs where it could occur. The second program used to probe for this error was:

```
PROGRAM B2;
VAR Even,Odd: INTEGER;
BEGIN
     WRITELN('Enter two numbers');
     READLN(Odd,Even);
END.
[ 2 3 ]
```

Ten students read 3 into "odd" and 2 into "even".

Order of declaration determines the order of reading

This was a fairly frequent error and it showed up with:

```
PROGRAM B4;
VAR A,B,C: INTEGER;
BEGIN
     WRITELN('Enter three numbers');
     READLN(C,B,A)
END.
[ 15 25 20 ]
```

These students argued that A was assigned the first number, B the second number and C the third number "because of the order the variables were declared". Typically the interviewer then modified the order of the variable declarations, and asked the students to rework the task. In all cases the response was consistent with this error.

Multiple-values read into a variable

This frequently occuring error was noted in:

```
PROGRAM B3;
VAR Even,Odd: INTEGER;
BEGIN
     WRITELN('Enter data: ');
     READLN(Even,Odd)
END.
[ 3 2 10 5 ]
```

These students consistently and confidently said that "even" was assigned the values 2 and 10 and "odd" 3 and 5. The data set was then frequently added to and the students continued to manifest this error. Further, when multiple-valued variables occur in a conditional statement these students either said that the _first_ value is used for comparison or that the comparison cannot be made or the program loops until the values in the variables have been "used".

2.2 Difficulties with print statements

The following three errors occurred occasionally with WRITELN statements:

a) WRITELN('Enter a number: ') caused a number to be read; similarly WRITELN('Enter 4 numbers: ') caused 4 numbers to be read.

b) WRITELN('Enter a number: ') caused the variable name _and_ its value to be printed.

c) After this statement has been executed the program can _choose_ a number from the data statement.

After we had encountered these students the following diagnostic sequence was devised:

```
X:= 3;
WRITELN('The value of X is 1');
WRITELN(X);
```

All subsequent students who did not have the errors noted above were able to cope with this correctly, but we are confident that the students with this error would have given the answer "1". This item has been added to our repetoire and will be used subsequently.

2.3 Assignment statements

ERIC

The first item of the screening test was designed to detect difficulties with assignment statements and supplementary examples produced for the interviews probe this construct further. Although the several errors only occurred occasionally, a total of nine students had difficulties with assignment statements. The difficulties noted include:

1. A:= B was interpreted as switching variables, A:=B <u>and</u> B:=A (3 students showed this error).

2. The assignment statement causes the instantiated statement to be printed. Given the sequence A:=2; B:=3; A:=B; one student said the computer would print 2 = 3.

3. The assignment statement had <u>no</u> effect (noted with 3 students).

4. A:= B was interpreted as A = B by 2 students.

2.4 Variables

The most significant "variable" error was the previously mentioned multiple value error. The following errors have also been noted occasionally:

1. Confusion of variables. In the sequence:

   READLN(P); Q:= Q+1;

   the latter statement is executed as if it were:

   Q:= P+1;

2. Values of variables are printed when the variable is encountered on the LHS (left-hand side) of an expression.

3. The value of the LHS variable is printed whenever its value changes.

2.5 Difficulties with loop constructions

A. Errors Common to both the FOR and WHILE constructs:

1. WRITELN adjacent to the loop is included in it, [IIIA2.1]. This error occurred frequently and was noted with nearly half the students interviewed. The programs which highlight this are:

```
PROGRAM G3;
VAR  P,Q: INTEGER;
BEGIN
     Q:= 0;
     WRITE('Enter a number: ');
     READLN(P);
     WHILE P <> 0 DO
       BEGIN
         IF P > 0 THEN
           Q:=Q + 1;
         WRITE('Enter a number: ');
         READLN(P)
     END;
     WRITELN(Q)
END.
[ 1 -1 -3 2 4 0 ]
```

and

```
PROGRAM A5;
VAR  I,X: INTEGER;
BEGIN
     FOR I:= 1 TO 3 DO
     BEGIN
        WRITELN(' Enter a number.');
        READLN(X)
     END;
     WRITELN(X)
END.
[ 6 3 4 2 4 1 8 ]
```

Curiously enough, those students who consistently make this error with the WHILE problem do not necessarily make it with the FOR loop, and vice versa.

2. Data-driven Looping

Several students indicated that the number of numbers in the data determined the times a loop was executed. Thus given the program:

```
PROGRAM A2;
VAR I, X: INTEGER;
BEGIN
     FOR I:=1 TO 3 DO
        BEGIN
           WRITELN('Enter a number.');
           READLN(X);
           WRITELN(X)
        END
END.
[ 6 3 4 2 4 1 8 ]
```

We have observed the following output:

```
6 3 4 2 4 1 8
6 3 4 2 4 1 8
6 3 4 2 4 1 8
```

The students explained that the number of values in the data set determined the number of columns, and the value of the FOR-loop limit (in this case 3) determined how many times the process was repeated.

Given 6 2 as input data this same student produced the following output:

```
6 2
6 2
6 2
```

3. Scope problems

Several errors involved misconceptions about which statements are repeated in loops and where loops begin and end.

a) Only the last instruction of a loop is executed multiple times. The other instructions are only executed once but the last instruction is executed the correct number of times. This error was noticed fairly frequently, but it only occurred in a loop where a WRITELN statement was the last one in the loop (thus it may be this error was caused by the write statement and not the loop construct).

b) BEGIN/END defines a loop. Two students thought that all the numbers in the data set would be printed despite the absence of a FOR or WHILE statement. The program is:

```
PROGRAM A3;
VAR  X: INTEGER;
BEGIN
     WRITELN('Enter a number.');
     READLN(X);
     WRITELN(X)
END.
[ 6 3 4 2 4 1 8 ]
```

T a variant on this error, scope of the loop is determined by indentation. In the case of some other programs, several students said that the WRITELN "went together with the FOR loop because they were lined up". One such program is:

ERIC

```
PROGRAM D1:
VAR  R, C: INTEGER;
BEGIN
    FOR R:= 1 TO 4 DO
       BEGIN
          FOR C:= 1 TO 3 DO
                 WRITE ('#');
          WRITELN
       END
END.
```

c) After a loop is executed control goes to the first statement of the program. This error was seen occasionally and in short programs could be interpreted as re-initializing variables each loop-cycle.

Although each of the scope errors in loops only arose occasionally, the total number of students who had difficulties with the scope concept was approximately one third of those interviewed.

B. Errors specific to FOR loops

We will just note 2 additional errors.

1. The control variable does not have a value inside the loop. (This occurred fairly frequently.)

2. The FOR loop statement acted like a constraint on the embedded READLN statement. A student said that only the numbers 3, 2, and 1 could be read with:

```
PROGRAM A5;
VAR  I,X: INTEGER;
BEGIN
     FOR I:= 1 TO 2 DO
     BEGIN
        WRITELN('Enter a number.');
        READLN(X)
     END;
     WRITELN(X)
END.
[ 6 3 4 2 4 1 8 ]
```

Although with this group of Pascal students this error only occurred once, we have previously noted it occurred frequently with students learning introductory BASIC.

2.6 Errors noted with IF statements

Four types of errors were noted occasionally with IF statements. However, 8 students (or 25%) made at least one of the errors:

1. Program execution is halted if the condi.ion is false and there is no ELSE branch.

2. Both the THEN and the ELSE branches are executed.

3. The THEN-statement is execu*ed whether or not the CONDITION is true.

4. IF <a> THEN <b>; <c>; is interpreted as

   IF <a> THEN <b> ELSE <c>;

2.7 Errors with procedures

These errors fell into two categories:

1. All statements including those in procedure bodies were executed in the order they appeared. This was a frequent error.

2. A fairly frequently occurring variant is that procedures are executed when they are encountered in a top-to-bottom scan of the program text and again when they are called.

2.8 Tracing and Debugging

As noted earlier tracing and debugging were not emphasized in this study, but we did include a program in the screening test and a program in the subsidary material which highlighted these issues. Further, interviewers frequently asked students to trace some of the other programs if they thought this would help determine the nature of the students' difficulties. From this activity, the several interviewers concluded that at least half of the students could not trace through programs systematically. Further, we concluded that these students often decided what the program should do on the basis of a few key statements, and would then "project this insight" onto the program as a whole.

1. Some students' interpretation of the following program is dominated by the variable Smallest -- "obviously this program is to find the smallest of a set of numbers". Thus this program highlights a variant of the semantic read misconception:

```
PROGRAM I1;
VAR  Smallest, Number: INTEGER;
BEGIN
     WRITELN('Enter a number: ');
     READLN(Number);
     Smallest:= Number;
     WHILE Smallest <> 0 DO
        BEGIN
           IF Smallest > Number THEN
           Smallest:= Number;
           WRITELN('Enter a number: ');
           READLN(Number)
        END;
     WRITELN(Smallest)
END.
[ 9 5 6 2 0 ]
```

These student assume that the first READLN(Number) statement reads the lowest value from the data line because, the smallest number is needed in the next statement, Smallest:=Number. This error was noted occasionally.

2. Students' interpretation of several programs relied on what would be reasonable output, rather than the actual output statements in the program:

```
PROGRAM F2;
VAR Number: INTEGER;
BEGIN
     WRITE('Enter a Number: ' );
     READLN(Number);
     IF Number = 7 THEN
       WRITELN('Unlucky Number');
     IF Number = 10 THEN
       WRITELN('Lucky Number');
     WRITELN('The Number was', Number)
END.
```

[4] [10] [7]

For instance when the number 10 was read, students said something like "Well it will print LUCKY NUMBER and that's all as there's no point doing the next line as we know the value must be 10 as it's a lucky number", an analogous explanation was given for the UNLUCKY number. These explanations were encountered frequently.

3. Summary of the data

The data is both rich and complex and so we shall attempt several overviews each of which will highlight some aspect.

Figure 1

Summary of Investigators' Assessment of Students Interviewed

	Male	Female	Total
No difficulties	3 (8.6%)	0 (0%)	3 (8.6%)
Minor difficulties	6 (17.14%)	8 (22.9%)	14 (40%)
Major difficulties	12 (34.28%)	6 (17.14%)	18 (51.4%)
	21	14	35

3.1 Summary of all the students interviewed

Incorrect variants of virtually every construct in the Pascal language were found with these students. At least 32 students out of the total population of 68 students had minor problems, and 18 of them (or 27%) had major difficulties. Anecdotally, teachers report that students debug programs by a "trial-and-error" method. This study lends support to this view, as, in the case of many students, several Pascal constructs are only partially understood (or subject to multiple interpretations), thus making a trial-and-error approach the only realistic alternative!

This study shows that even after a full semester of Pascal, students' knowledge of the conceptual machine underlying Pascal can be very fuzzy. This is a more widespread problem than we had expected and one which is not totally appreciated by the teachers who frequently set a performance- based completion criteria for the class. Not unreasonably, programming tasks are completed jointly by several students often masking the several misunderstandings of the individual students.

Each interviewer classified each student's performance as having essentially no difficulties, minor- and major-difficulties; this information is tabulated in figure 1. From this figure we note that 3 of the students did not have any problem in the interview although the screening test indicates they had. In most cases this has been attributed to the students rushing the test or not taking it seriously. (Note: all 3 were males). Of the remaining students, the figures show they fall nearly equally between those who have minor and major difficulties. However there is a greater proportion of males who had major difficulties than females, namely 12/18 males (66.6%) had major difficulties as compared with 6/14 (42.9%) females. (These figures are based on the population interviewed and not the total numbers screened). Interestingly, our assessment of the students closely with the teacher's evaluation.

3.2 Profiles of typical students

So far in this paper we have given an account of the errors which have been noted in the population with some indication of the number of students who manifested that error. By way of contrast, in this section we will describe all the difficulties noted for two students; one was described as having minor difficulties and the other having major difficulties.

Example of a student with minor difficulties.

This student appeared to have two errors:

1. Assignment being interpreted as a switch

   A:= B is interpreted as A:= B and B:= A.

   (The student did not manifest this behavior consistently).

2. Statements in procedures are executed as they are encountered.

   (The student appeared to be consistent with this error).

Note too that when we interviewed this class, procedures had only recently been introduced.

Example of a student with major difficulties.

The errors reported with this student were:

1. Semantically constrained reads (a consistent error).
2. The alphabetic ordering of the variables determines the order in which the data is read (not applied consistently).
3. Read a value when a variable is encountered in a statement (not used consistently).
4. WRITELN('Enter a Number') causes a value to be read (not used consistently).

5. WRITELN adjacent to a loop construct is considered a part of the loop (a consistent error).

6. The number of data elements determines the number of times a loop is executed (a consistent error).

7. The control variable does not have a value in the loop body.

8. The order of execution of 2 statements was inverted consistently in one program:

```
PROGRAM G1;
VAR  Number: INTEGER;
BEGIN
     Number:= 0;
     WHILE Number < 5 DO
     BEGIN
         WRITELN(Number);
         Number:=Number + 1
     END
END.
```

was executed as if the loop body was:

```
     Number:= Number + 1; WRITELN(Number);
```

However, this behaviour was <u>not</u> noted with other programs.

9. Infer program function based on a number of commands.

A section of the interview with this student is quoted in section 2.8 where this error is described. Essentially we believe this student's interpretation of the program was dominated by several key statements in the code, and the notion that the machine would act "reasonably". Why, he argued, would the machine execute the last WRITELN which gives you the value of the number, when it has already told you that the number was lucky and so you'd know it had the value of 10?

We have classified this student as having major difficulties not only because of the sizeable number of difficulties but because we believe some of the misconceptions will be hard to remediate. For example, we believe that the last 2 errors (8 and 9) will be difficult to remediate as here the student is calling upon a lot of common-sense knowledge. This issue will be discussed again in section 3.3.

3.3 Error classification

Implied at the end of the last subsection is the belief that some misconceptions will be much easier to remediate than others. In the algebra domain, for instance, Sleeman has suggested that some errors occur because the user omits one of the substeps of a rule which he essentially knows; we have called these manipulative errors (Sleeman, in press). Other errors indicated that the student has little understanding of basic concepts in the algebra domain. For example we have seen the expression

3 X + 4 X = 19 changed to X + X = 19 -3 - 4

We have referred to errors of this sort as being parsing errors. To generalize this classification to programming, and possibly to other domains, we propose referring to these classes of errors as surface and deep errors.[7] We suggest that an example of a surface error in this domain is seeing the FOR loop with range 0 to 5 as 1 to 5. Lack of understanding of variables is a complex issue[8] and we suggest should be classed as a deep error. As indicated earlier we believe that the inference of the functionality of a program from a few key instructions is a deep error and one which arises from the user bringing common sense reasoning to bear on a formally defined domain. In addition to this issue we believe that many of the errors noted, for example, semantically constrained reads can be explained by the user attributing to the machine the reasoning power of an average human. We refer to this subclass of errors as the "reasonably human" error class.

4. Comparison with the difficulties noted in this and other studies

Finally we should recall that for the most part we have presented students with syntactically correct programs hence the significant difference in these findings and those of Ripley and Druseikis, 1978. However, we noticed that these students had considerable problems with the notion of scope (which supports Ripley & Druseikis's third observation, see section 1). Moreover, we generally noticed that the students we worked with had only a fuzzy idea of syntax, and we would not be surprised if they made the types of punctuation errors noted in the earlier study in their own programs.

With our students, only one student out of the 35 interviewed treated the WHILE statement as a demon, a marked contrast to the Yale data.

5. FURTHER WORK

We plan to:

1. look at the difficulties high-school students have with more advanced concepts of Pascal (in this study we only touched upon procedures);
2. investigate the difficulties which high-school students have with LOGO (to give us a third point of reference);
3. determine whether it is possible to remediate the sort of errors we noted in this study. If this is possible, try different remedial strategies to determine their effectiveness;
4. speculate further about how such misunderstandings arise and possibly experiment with different teaching/presentation strategies.

6. ACKNOWLEDGEMENTS

We wish to thank the teacher and students who participated in this study. Haym Hirsch and Alan Char helped with the interviewing. Marcia Linn helped by providing access to an earlier test of BASIC programming. We also thank the Study of Stanford and the Schools, directed by Dean M. Atkin and President Kennedy, and D. Walker and M. Lepper (Chairs of the Panel on Education and Technology) for providing funding for the study. Finally, Mrs. Dorothy Brink provided invaluable secretarial support.

7. REFERENCES

Bonar, J. & Soloway, E. (1982). Uncovering principles & novice programming. Research Report 240, Dept. of Computer Science, Yale University

Bork, A. M. (1971). Learning to program for the science student. J. Educ. Data Processing, 8, pp. 1-5.

Davis, R. B., Jockusch, E. & McKnight, C. (1978). Cognitive Processes in Learning Algebra. J. of Children's Mathematical Behavior, 2, pp. 1-320.

Du Boulay, B. & O'Shea, T. (1981). Teaching novices programming. In M. Coombs & J. Alty (Eds.), Computing skills and the user interface London: Academic Press./pp. 147-200.

Gentner, D. & Stevens, A. L. (Eds.). (1983). Mental Models. Hillsdale, NJ: Erlbaum.

Kuechemann, D. (1981). Algebra: In Children's Understanding of Mathematics: 11-16 ed. K. M. Hart: Murray: London, pp. 102-119.

Larkin, J. H., McDermott, J., Simon, D.P. & Simon, H.A. (1980). Models of competence in solving phys[illegible] problems. Cognitive Science, 4, pp. 317-345.

Lemos, R. S. (1975). FORTRAN programming: an analysis of pedagogical alternatives. J. of Educ. Data Processing. 12. pp. 21-29.

Norman, D. A. (1983). Some observations on mental models in Mental Models edited by Gentner, D & Stevents, A. Erlbaum.

Putnam, R. T., ''leeman, D., Baxter, J. A. & Kuspa, L. K. (in press). A Summary of Misconceptions of High School BASIC Programmers.

Ripley, G. D. & Druseikis, F. C. (1978). A statistical analysis of syntax errors. Computer Languages, 3, pp. 227-240.

Sheil, B. The psychological study of programming. (1981) Computing Surveys, 13, pp. 101-120.

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. Int. J. Computer Information Science, 5, pp. 123-143.

Sleeman, D. (in press) An Attempt to understand student's understanding of basic algebra. Cognitive Science.

Sleeman, D. (in preparation). Protocol Analysis & Interviewing: some methodological issues.

Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. (1982). What do novices know about programming? In A. Badre and B. Shneiderman (Eds.), Directions in human/computer interaction Norwood, NJ: Ablex. pp 27-54.

Wegner, P. (1968). Programming Languages: Information Structures and Machine Organization. McGraw Hill, pp. 84-91.

Footnotes

1. Also Computer Science Department.

2. Lemos, (1975) claims that the approach of analyzing complete programs was no more effective than the traditional one -- however, we feel that this important line of investigation should not be abandoned because of a single negative data point.

3. That is given a WHILE statement of the form:

```
WHILE cond DO
  BEGIN S1;
          ...
          Sn
END
```

after executing a statement Si, of the WHILE body they would check to see whether the COND is still true, if not they would skip the remaining statements.

4. Most students had some prior exposure to BASIC; the effect of a previous programming language on a second language is an issue to be considered in a further study.

5. It is difficult to categorize unambiguously many of the errors noted. For example: is the last error mentioned a "READ" or a "variable" error?

ERIC

In this paper we make an arbitrary assignment to a class which seems reasonable; in some cases we discuss possible alternative categories.

6. The convention used in this and subsequent programs is that the data is provided in brackets immediately following the program; multiple brackets indicates multiple sets of data.

7. Following the recent distinctions introduced to designate the level of an expert system's knowledge of its domain.

8. Also encountered in algebra (Kuechemann, 1981).